An XML Standards Based Authorization Framework for Mobile Agents

G. Navarro and J. Borrell

Dept. of Information and Communications Engineering
Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain
{gnavarro, jborrell}@ccd.uab.es

Abstract. An outstanding security problem in mobile agent systems is resource access control, or authorization in its broader sense. In this paper we present an authorization framework for mobile agents. The system takes as a base distributed RBAC policies allowing the discretionary delegation of authorizations. A solution is provided to assign authorizations to mobile agents in a safe manner. Mobile agents do not need to carry sensitive information such as private keys nor they have to perform sensitive cryptographic operations. The proposed framework makes extensive use of security standards, introducing XACML and SAML in mobile agent system. These are widely accepted standards currently used in Web Services and Grid.

Keywords: Mobile Agents, Authorization, Access Control, XACML, SAML.

1 Introduction

During the last years, mobile agent technologies have witnessed an steady, if not fast, increase in popularity. Probably, the main hurdle to a wider adoption are the security issues that mobility brings to the picture [7]. Among them, an outstanding one is resource access control. Traditional access control methods rely on the use of centralized solutions based on the authentication of global identities (for example, via X.509 certificates). These methods allow to explicitly limit access to a given resource through attribute certificates or Access Control Lists, and rely on a centralized control via a single authority. Despite providing effective means of protection, these techniques suffer from serious drawbacks; in particular, they give raise to closed and hardly scalable systems. Practical mobile agent systems demand lightweight, flexible and scalable solutions for access control, in order to cope with the highly heterogeneous nature of their clients. In the same vein, solutions depending on centralized entities (such as traditional Certification Authorities) should be avoided.

Recent developments in the area of access control, in an attempt to further ease access control management, have brought into the picture Role-based Access Control (RBAC) [11]. In these schemes, privileges of principals requesting access to a resource are determined by their membership to predefined roles. The use of RBAC greatly simplifies access control management and is specially suited to mobile agents scenarios, where agents privileges are subsumed in a possibly more general RBAC system.

Even in RBAC environments, there may be some situations where more flexibility is required. Discretionary delegation of authorizations between users may provide flexibility beyond RBAC policies. A user may temporary delegate some rights to another one without having to modify the system policies. Delegation of authorizations has been successfully introduced by *trust management* systems such as *Simple Public Key Infrastructure/Simple Distributed Secure Infrastructure* (SPKI/SDSI) [9] and KeyNote [3].

This article presents an access control framework for mobile agents. We combine RBAC and discretionary delegation of authorizations to provide a flexible, and distributed system for access control in such scenarios. In our approach, mobile agents do not carry explicit information regarding resources access control, avoiding the privacy concerns associated with sensitive data embedded in mobile code. We have implemented our proposed scheme on MARISM-A, a JADE-based agent platform. Our framework is called XMAS (*XML-based Mobile agents Authorization System*).

In Section 2 we explain the motivations behind XMAS. Sections 3 and 4 describe the main naming and authorization issues. We describe the main components and functionality of XMAS in Section 5. Section 6 details how roles are assigned to mobile agents. Finally, Section 7 summarizes our conclusions.

2 Motivations and Related Work

Proposals for access control in multiagent systems supporting agent mobility are scanty. Most of the security work in mobile agents deals with protecting communications, itineraries, and so on, but few of them deal with the protection of resources and the management of access control rights.

Usually, proposed systems rely on ad-hoc and centralized solutions. As an example, in [23] the authors propose a solution based on proprietary credentials, which are carried by the agent as authorizations. These credentials are combined with the Java protection domains to provide resource access control. The solution is interesting but it relies too much on the platform itself, and more precisely in Java. The system cannot be applied to other platforms and it is not distributed, since it requires a centralized policy. A similar approach was adopted in [21]. And most notably [14].

JADE (`http://jade.tilab.it`) also provides a security add-on[13], which is also based on the Java security architecture and does not support agent mobility. FIPA (Foundation for Intelligent Physical Agents: `http://fipa.org`) also began to consider agent security through a technical committee, but the work has not been finished, and there is no FIPA recommendation at the moment on security related topics.

Other mobile agent platforms, such as NOMADS [21], provide access control based on Java-specific solutions and eventually used KAoS to provide high level policy languages and tools. KAoS [4] is a high level policy language currently focused on semantic web services. Despite its complexity, KAoS does not support mobile agents.

On the other hand, several communities have substantially contributed to access control and authorization management in distributed environments, most notably in Web Services and Grid. Systems such as Shibboleth [10], PRIMA [16], PERMIS [6], or Cardea [15], are just an example. These systems tend to be policy-based and provide support for Role-based Access Control (RBAC).

Another important initiative has been the development of standards for expressing authorization information. For instances the *eXtensible Access Control Markup Language* (XACML)[22] is a general purpose access control policy language specified in XML. XACML is intended to accommodate a wide variety of applications and environments. Together with the policy specification language it also provides a query and response format for authorization decision requests. The expressiveness, semantics for determining policy applicability, support for advanced features, and the fact that it was designed with distributed environments in mind, makes XACML a great standardized base for large-scale authorization frameworks.

At the same time, the *Secure Assertion Markup Language* (SAML)[19], provides a standard XML-based framework for exchanging security information between online business partners. Security information is exchanged in form of *assertions*. SAML provides three types of *assertion statements*: Authentication, Attribute, and AuthorizationDecision. Broadly speaking an assertion has an *issuer*, a *subject* or *subjects*, some *conditions* that express the validity specification of the assertion, and the *statement*. The assertion may be signed by the issuer. SAML also provides query/response protocols to exchange assertions and bindings over SOAP and HTTP.

XACML and SAML are being widely adopted by the previous projects in web and grid services, as well as other industry solutions (Liberty Alliance, for instance). These two standards are also well suited to coexists together. For instance, Cardea uses SAML protocols in an XACML policy based environment. In XMAS, we have choose to adopt a combination of XACML and SAML. XACML is used to express the main authorization policy, which is an RBAC policy. While SAML is used to express discretionary delegation statements and provides the protocols to exchange all the information.

This work is the continuation of a previous framework for authorization management in mobile agent systems [17], based on SPKI/SDSI. From this previous experience, we have developed a more scalable system making use of current standards and introducing a more powerful delegation model, among other things.

The main contributions of XMAS is to provide a novel authorization framework for mobile agents. It presents the combination of RBAC with discretionary delegation of permissions or authorizations. The proposed use of XACML and SAML standards in mobile agent system, will allow agents to interact with other systems such as Web Services or Grids. We also describe a safe way to assign roles to mobile agents, so the mobile agent does not need to carry sensitive information.

3 Naming Schema

XMAS uses a distributed naming schema strongly influenced by SPKI/SDSI, which is used to create and manage roles and groups. Each entity, (agent, platform, human user, etc.) is denoted as a *principal*. All principals (including static agents) have a pair of cryptographic keys, except mobile agents. The public key acts as a global identifier of the principal. In order to make it more manageable one can use the *hash* of the public key as an abbreviation for the public key. Each principal generates by itself the keys and is responsible of its own identity, so there is no need for a centralized CA, although it can be used if it is needed. Mobile agents are identified by a hash of its code.

A principal can define local names of entities under its own responsibility. In order to do that, an entity has an associated local name space called *name container*. The name container has entries of the type: `(<principal>,<local-name>)`. The *principal* corresponds to the principal for whom the local name is being defined, and *local-name* is an arbitrary string. The *principal* can be specified as a public key or as a *fully qualified name* (see below).

For example, consider a principal with public key PK_0, which creates an agent with public key PK_1 and wants to name it *my-agent*. The name container of the entity will have an entry of the form: (PK_1, `my-agent`). Now on, the agent PK_1 can be referenced by the name *my-agent* in the local name space of PK_0. An important issues is that a third party can make a reference to a name defined in other name containers through a *fully qualified name*. A name container is identified by the public key of the owner, so the fully qualified name PK_0 *my-agent* makes reference to the name *my-agent* defined in the name container of PK_0. It is important to note that given the properties of cryptographic keys, it is commonly assumed the uniqueness of the public key, so fully qualified names are globally unique.

These names, make it very easy for a principal to create groups or roles. For instances, a user PK_{admin} can create a group *employees* with members PK_a, PK_b and the agent PK_1, by adding the following entries in its name container:

$$(PK_a,\ \textit{employee})$$
$$(PK_b,\ \textit{employee})$$
$$(PK_0\ \textit{my-agent},\ \textit{employee})$$

In our framework names are expressed not only as identifiers of a principal but also as *attributes* in the case of roles. A name container entry can be expressed as a SAML assertion, where the issuer is the owner of the name container, the subject is the principal and the name is expressed as an AttributeStatement. We denote such an assertion as:

$$\{(PK_{admin}\ \textit{security-staff},\ \textit{employee})\}_{PK^{-1}_{admin}}$$

where PK^{-1}_{admin} denotes the private key corresponding to the public key PK_{admin}, which digitally sings the assertion determining the issuer or the owner of the name container where the name is defined (note that this assertion can only be issued by the owner of the container). The assertion may also contain validity conditions, which are not shown for clarity reasons.

Although any principal may create and manage roles for its own purpose, in order to make them available to the XMAS framework, they need to be introduced into an XACML policy managed by a special entity, the Role Manager (see Section 5.2).

4 Delegation of Authorization

Authorizations may be assigned to principals either through an XACML policy rule or through a SAML assertion. An authorization will have an issuer, the principal granting the authorization, which in the case of an XACML rule will be the policy owner, and in the case of a SAML assertion will be the issuer of the assertion. It also has a subject, and the authorization itself.

A key point of XMAS is the ability to allow *delegation* of authorizations between principals. Furthermore, we adopt a *constrained delegation model*, which provides a powerful delegation mechanism. It allows to differentiate between the right to delegate an authorization and the authorization itself. That is, a principal can have the authority to delegate a given authorization but may not be able to hold (and use) the authorization. This is different from the approach in *trust management* systems [3,9] in which the right to delegate a privilege can be given only to those that have the privilege for themselves. A formal definition and a full description of the constrained delegation model can be found in [2,20].

In short, there are two types of authorizations:

- *Access-level authorization*: defines an access-level permission such as *read file*, assigned to a principal. It is denoted as: $authz(s, a)$, where s is the subject (a principal) and a is the specific access-level permission.
- *Management-level authorization*: defines the authority to declare an access-level authorization. That is, the right to delegate an access-level authorization. We denote it as: $pow(s, \phi)$, where s is the subject (a principal) and ϕ is either an access-level authorization or another management-level authorization.

A complete authorization can be denoted as:

$$\{(K_s,\ p)\}_{K_i^{-1}}$$

Where the subject K_s receives the authorization p from the issuer K_i. Again, K_i^{-1} denotes the private key associated with K_i, which issues the authorization. Both names and authorizations are denoted with a 2-tuple, the main difference between them being the second element. In the case of an authorization it will be of the form of: $auth(\ldots)$, or $pow(\ldots)$.

The flexibility and the power introduced by this constrained delegation model comes with a cost. In order to determine if a principal has gained an authorization through delegation, we have to find a delegation chain from a source of authority to the principal. Finding this *authorization proof* may become very complex. In order to simplify it we rely on a combination a *privilege calculus* [20], to find authorization proofs in the presence of *constrained delegation*, and the name resolution and reduction rules of the SPKI/SDSI certificate chain discovery algorithm[8]. Broadly speaking, in order to find delegation chains, we use the name resolution algorithms to resolve all the names into public keys (or hashes in the case of mobile agents), and then apply the privilege calculus to find the authorization proof. The procedure is computationally feasible presenting a polynomial order of complexity, an it is performed by an special entity, the delegation Assertion Repository Manager (see Section 5.4).

5 XMAS Components

The XMAS system is implemented on top of MARISM-A [18], a secure mobile agent platform implemented in Java. It provides extensions on top of the JADE system. JADE implements the standard elements of the FIPA specification and provides additional services for the management of the platform. Mobility is achieved by the MARISM-A mobility component which is integrated into JADE. On top of JADE there are the main MARISM-A components such as the authorization framework presented in this paper, and other MARISM-A services such as: cryptographic service, directory service, service discovery, etc.

Agents in MARISM-A can be mobile or static, depending on the need of the agent to visit other agencies to fulfill its task. There are several types of mobile agents according to the characteristics of its architecture: basic or recursive structure, plain or encrypted, itinerary representation method, etc. Agents can communicate with each other through the agency communication service.

XMAS is made up of five independent components, which interact to perform all the required functionality. This components are implemented as static agents. The messages exchanged between these components are SAML protocol messages, enclosed in FIPA Agent Communication Language, and using FIPA ontologies. The SAML protocols already provide bindings for transport over SOAP and HTTP, which are used to communicate XMAS components with entities outside the multi-agent domain, such as external authorities, normally in the form of web services or grid services, where the use of SAML and SOAP is widespread. The reason why we choose FIPA ACL is that it has become an standard in multi-agent environments. Most of the existing agents platforms support the FIPA specifications, which provide transport over IIOP, HTTP, and WAP.

In order to locate required information and modules. XMAS relies on a service discovery infrastructure. It is provided by the underlying agent-platform implementing the FIPA agent discovery specification [12]. This is specially relevant in the presence of mobile agents because an authorization decision may involve the gathering of information from components located in different platforms. In the case of XACML, the XACML policies already provide mechanisms to easily distribute policies through references, without the need for discovery services. But for example, the delegation chain discovery can use the service to locate an specific authorization manager.

5.1 Authorization Manager (AM)

The *Authorization Manager* (AM) manages the assignment of a set of specific authorizations to specific roles. It may also provide the ability to delegate authorizations to other AMs in order to distribute the authorization management. Since the authorization policy is local to the AM agent, it does not need to follow any specification and its format could be implementation-dependent. Even so, we use XACML policies to provide a standardized, and uniform approach.

The AM has two different local policies, expressed in XACML. The first one is the *XML Authorization Policy* (XAP). The XAP specifies the authorization assignment to roles. In other words determine, which roles hold which permissions or authorizations. The second one is the *Administrative XML Authorization Policy* (XAP-*adm*). The XAP-*adm* determines the administrative authorities for the authorizations managed by the AM. An AM may receive a request to issue an authorization for an specific role, if the authorization comes (directly or indirectly) from one of the authorities for that specific authorization, it is granted and the XAP is modified accordingly.

Principals sending requests to the AM will normally be other agents using SAML protocols. Even so, we have provided a GUI tool for the easy specification of the AM policies by human administrators: the *AM-console*.

As Figure 1 shows, the XAP is possibly the most complex policy of the system. Originally we used a proprietary definition of the XAP policy, but we have recently adopted the proposal on RBAC profile for XACML [1]. It is currently a Committee Draft in the XACML version 2.0 specification.

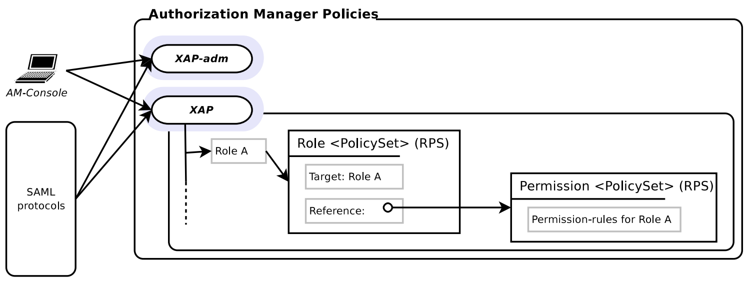

Fig. 1. Authorization Manager (AM) policies

The XAP is made up of several *<PolicySet>s*. For each role there is a *Role <PolicySet>* (RPS) associating the role to a *Permission <PolicySet>* (PPS), which determines the actual permissions associated with the role. It is important to note that the entry point to query the XAP is the RPS and never the PPS. Or in other word, a PPS cannot be directly consulted, the only way to access it is through an RPS. This allows to support role hierarchies, ensuring that only principals of the given role can gain access to the permissions in the given PPS. Role hierarchy is thus defined by including a reference to the PPS associated with the junior (or sub) role in the PPS of the senior (or super) role.

5.2 Role Manager (RM)

The *Role Manager* (RM) manages a set of roles, mainly role membership (Section 6 details how roles are assigned to mobile agents).It has an *XML Role Policy* (XRP) and an *Administrative XML Role Policy* (XRP-*adm*). The XRP-*adm* policy specifies administrative role authorities. That is, principals that can request the creation of roles, the assignment of principals to roles, or the specification of constraints on the role assignment.

The XRP policy has two types of XACML policies. It has policies to define role membership and a *Role Assignment Policy*. The role membership is expressed as an XACML Attribute, associated to the given principal (see Section 3). On the other hand, the Role Assignment Policy, is used to answer the question “Is subject X allowed to have role R_i enabled?”[1].

As in the case of the AM, there is a GUI tool (*RM-console*) specially designed so RM policies can be managed by a human, although they are mainly intended to be managed by autonomous agents through SAML protocols.

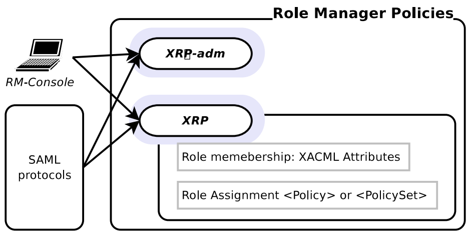
Role Manager Policies
RM-Console
XRP-adm
XRP
SAML protocols
Role memebership: XACML Attributes
Role Assignment <Policy> or <PolicySet>

Fig. 2. Role Manager (RM) policies

5.3 Resource Controller (RC)

The *Resource Controller* (RC) main task is to control the access to a set of resource. It acts as a Policy Enforcement Point [24], receives access requests from principals and allows or denies the request depending on authorization responses from the Authorization Decision Engine.

5.4 Delegation Assertion Repository Manager (ARM)

In XMAS, principals may delegate authorizations in a discretionary way as SAML assertions. This assertions need to be gathered to perform the authorization decision. Although each principal can maintain a local cache of issued assertions, having to search and query all principals for assertions relating a given request can be too expensive in terms of communications due to the high complexity and possibilities imposed by possible delegation paths.

The *delegation Assertion Repository Manager* (ARM), keeps and updated repository of SAML delegation assertions. When a principal issues a delegation assertion it stores the assertion in the repository. Furthermore the ARM can find partial *authorization proofs* by finding delegation chains from the repository (see Section 4). This way we solve the problems derived from assertion distribution and leave the task to perform chain discoveries to the ARM and not to the other principals. It decreases communication traffic, assertions do not need to travel constantly from one principal to another, and reduces the task that generic principals need to perform.

5.5 Authorization Decision Engine (ADE)

The *Authorization Decision Engine* (ADE) is the main responsible for determining authorization decisions acting as a Policy Decision Point [24]. Given an authorization request, normally coming from an RC, the ADE is able to determine if the request has to be granted or denied by retrieving information from required AMs, RMs, ARMs, and other possible sources of information such as external Attribute Authorities.

An important issue with the ADE is that it is unique for a single agent platform. XMAS allows to place any number of AM, RM, ARM and RC in a single platform, they may be set up by different principals, or for different applications. But there is only one ADE in the platform. This ADE is the one used by all the RCs in the platform. Note that there is an implicit trust relation between the resources or service owners and the platform where they are placed.

Conflict Resolution. Delegation of authorizations was introduced to provide more flexibility in the system, but at the same time it may introduce possible inconsistencies. It is possible to find conflicts between the RBAC policies and the delegation assertions. For instance, we can find a principal who has received an authorization from a delegation assertion, but at the same time has enabled a role, which explicitly denies such authorization.

We adopt a *closed policy*, all authorizations are denied by default, and the policies express permitted authorizations. In XACML it is possible to specify deny rules, but this rules are normally used to express exceptions and are not used in a normal base. Even so, the ADE provides a conflict resolution procedure to resolve possible inconsistencies. The ADE can obtain three different evaluation results: an XACML deny or XACML permit from the RBAC policies, and an explicit permit from delegation assertions. The ADE uses the following order of precedence:

```
begin
    if (XACML-decision == Deny) then return Deny;
    if (XACML-decision == Permit) then return Permit;
    if (there is a delegation chain == Permit) then return Permit;
    else return Deny;
end
```

Algorithm 1. ADE combining algorithm

Note that negative rules from the RBAC policies supersede the delegation statements. It is possible to apply constraints on delegations in the RBAC policies. For example it is possible to define a deny rule in the RBAC policies for a role, so principals cannot delegate authorizations received as a result of being members of that role.

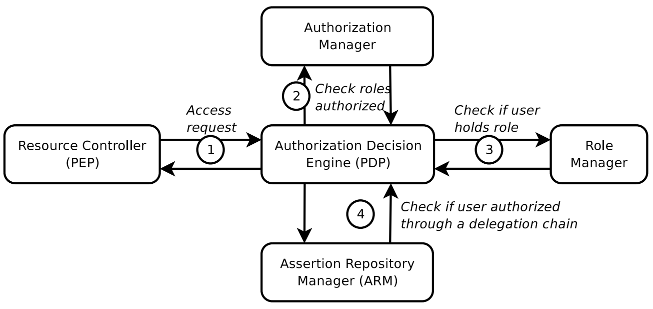
Authorization Manager
2
Check roles authorized
Access request
1
Resource Controller (PEP)
Authorization Decision Engine (PDP)
Check if user holds role
3
Role Manager
4
Check if user authorized through a delegation chain
Assertion Repository Manager (ARM)

Fig. 3. ADE data flow

Figure 3 shows a simplified data flow for the decision engine. Note that, for example, querying the AM to check the roles authorized, may involve the interaction with several AMs depending on the one that handles the specific authorization.

6 Establishing Mobile Agents Role Membership

One of the first problems we found when planning the authorization model, is if a mobile agent should have a cryptographic key pair and be considered as a principal. A mobile agent cannot trivially store a private key, as well as perform cryptographic operations such as digital signatures. There are some proposals to store sensitive information (private keys) in mobile agents [5]. But the problem arises when the mobile agent uses the private key to compute a cryptographic operation. The agency where the agent is in execution will be able to see the private key or at least, reproduce the operation. As a result we consider that a mobile agent should not have a private key.

Since mobile agents cannot have private keys, we can not delegate authorizations to a mobile agent or make it member of a role. Our approach is to set, as member of the role, a hash of the agent's code. A principal can be identified by a a public key or a hash of a public key. So a hash may be seen as a principal, subject of a certificate. This is even supported by certificate frameworks such as SPKI/SDSI or even X.509, where a hash of an object can be considered as a principal.

In order to establish the role membership of a mobile agent we consider two different approaches. To show them we use a simple example where a user K_u is member of the role $physician$ defined by a Role Manager RM_0 of the agency i, which has a resource controlled by a given RC. The role $physician$ allows its members to access the given resource in agency i. The user has a mobile agent with the code m_i to be executed in platform i. The goal is to set the hash of m_i as member of the role $physician$.

User-managed Role. RM_0 makes member of the role $physician$ a role (or group) defined by the user K_u, say $agent$. Then, the user K_u can make member of its role $agent$ any hash of agent's code:

$$\{(K_u \textit{ agent}, \textit{ physician})\}_{RM_0^{-1}}$$
$$\{(hash(m_i), \textit{ agent})\}_{K_u^{-1}}$$

RM-managed Role. RM_0 makes member of the role $physician$ the user K_u. Then the users sends a request to the RM_0 to set the agent code's hash as member of the role:

$$\{(K_u, \textit{ physician})\}_{RM_0^{-1}}$$
$$\{(hash(m_i), \textit{ physician})\}_{RM_0^{-1}}$$

The *user-managed role*, is quite straightforward, gives the user full flexibility to manage the role K_u $agent$, and does not require any special mechanism or protocol. The user may add to the role any agent (or even user) she wants. The main problem with this first approach is related to the accountability of the system. In e-commerce application there will be different degrees of trust between users. For example, a hospital may trust an internal physician to manage its role $agent$, but a client from an external research center may not be so trusted. In the first case we will use a *user-managed role*, while in the second one we will use an *RM-managed role*.

This last approach requires an additional protocol, which is implemented as a SAML protocol. The user sends an agent role assignment request to the RM, including m_i. The RM verifies the client's request, if permitted, it computes the hash of m_i and issues the corresponding XACML attribute, in its XRP.

The RM may store the code m_i to use it for further security audits (or a hash of the code). Note that this approach does not allow the user to manage the role and extend it to other agents. The user needs to send a request for each agent.

When the mobile agent arrives to agency i, it will send an access request to the RC controlling the resource. The RC just has to compute the hash of the code m_i and check, through a request to the ADE, if the agent is authorized to access the resource.

The main drawback of this approach is that a mobile agent is not capable of issuing a delegation assertions by itself, since the agent can not sign them. But note that this does not mean that the agent cannot issue or delegate an authorization, which may be certified by a trustee.

Authorizations associated to an agent are normally determined by its role membership. This way we can say that the agent will have *dynamically assigned authorizations* during its lifetime. If the authorizations associated with a role change, the authorizations related to the agent also change.

7 Conclusions

In this paper we have presented a novel authorization framework for mobile agent systems, XMAS. The main contributions in the field of mobile agents can be summarized in three points. First, it presents an RBAC-like distributed policy and at the same time adds support for discretionary delegation of authorizations. Second, the framework makes extensive use of current security standards for distributed systems, more specially XACML and SAML, which easies the interoperation with other systems such as Web Services or Grid. Finally we propose a mechanism to authorize and assign roles to mobile agents, so the agent does not need to carry sensitive information (such as private keys) or perform cryptographic operations. The proposed framework is currently implemented on top of the MARISM-A project, a secure mobile agent platform based on JADE.

The use of XACML has greatly contributed to provide a uniform and standardized manner to handle distributed RBAC policies. As a side effect we have find that XACML documents are quite complex and bloated, and cannot be manually edited by administrators without a good knowledge of the language. This has forced us to create graphical user interfaces to manipulate the policy documents.

Acknowledgments

The authors would like to thank Dr. Jose Antonio Ortega-Ruiz for his comments on previous drafts of the paper and for suggesting the name XMAS.

This work has been partially funded by the Spanish Ministry of Science and Technology (MCYT) though the project TIC2003-02041.

References

1. A. Anderson, ed. Core and Hierarchical Role Based Access Control (RBAC) profile of XACML, Version 2.0. OASIS XACML-TC, Committee Draft 01, September 2004.
2. O. Bandmann, M. Dam, and B. Sadighi-Firozabadi. Constrained delegation. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 131–140, Oakland, CA, May 2002. IEEE Computer Society Press.
3. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System. RFC 2704, IETF, September 1999.
4. J.M. Bradshaw, S. Dutfield, P. Benoit, and J.D. Woolley. KAoS: Toward an industrial-strength open agent architecture. Software Agents, 1997.
5. K. Cartrysse and J.C.A. van der Lubbe. Privacy in mobile agents. In *First IEEE Symposium on Multi-Agent Security and Survivability*, 2004.
6. David W. Chadwick and Alexander Otenko. The PERMIS X.509 role based privilege management infrastructure. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*. ACM Press, 2002.
7. D. Chess. Security issues of mobile agents. In *Mobile Agents*, volume 1477 of *LNCS*. Springer-Verlag, 1998.
8. D. Clarke, J. Elien, C. Ellison, M. Fredette, A. Morcos, and R. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(9):285–322, 2001.
9. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. RFC 2693: SPKI certificate theory. The Internet Society, September 1999.
10. M. Erdos and S. Cantor. Shibboleth architecture v05. Internet2/MACE, May 2002.
11. D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R Chandramouli. Proposed NIST standard for role-based access control. In *ACM Transactions on Information and System Security*, volume 4, 2001.
12. FIPA TC Ad Hoc. Fipa agent discovery service specification, November 2003.
13. JADE Board. Jade security guide. JADE-S Version 2 add-on, 2005.
14. G. Karjoth, D.B. Lange, and M. Oshima. *Mobile Agents and Security*, volume 1419 of *LNCS*, chapter A Security Model for Aglets. Springer-Verlag, 1998.
15. R. Lepro. Cardea: Dynamic access control in distributed systems. Technical report, NASA Advanced Supercomputing (NAS) Division, 2003.
16. M. Lorch, D. B. Adams, D. Kafura, M. S. R. Koneni, A. Rathi, and S. Shah. The prima system for privilege management, authorization and enforcement in grid environments. In *Fourth International Workshop on Grid Computing*, 2003.
17. G. Navarro, S. Robles, and J. Borrell. Role-based access control for e-commerce sea-of-data applications. In *Information Security Conference 2002*, September/October 2002.
18. S. Robles, J. Mir, J. Ametller, and J. Borrell. Implementation of Secure Architectures for Mobile Agents in MARISM-A. In *Fourth Int. Workshop on Mobile Agents for Telecommunication Applications*, 2002.
19. S. Cantor, J. Kemp, R. Philpott and E. Maler, ed. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS XACML-TC, Committee Draft 04, March 2005.
20. B. Sadighi-Firozabadi, M. Sergot, and O. Bandmann. Using authority certificates to create management structures. In *Proceedings of Security Protocols, 9th Internatinal Workshop*, April 2002.
21. N. Suri, J. Bradshaw, M. Breedya, P. Groth, G. Hill, R. Jeffers, and T. Mitrovich. An overview of the NOMADS mobile agent system. In *Proceedings of 14th European Conference on Object-Oriented Programming*, 2000.

22. T. Moses, ed. eXtensible Access Control Markup Language (XACML), Version 2.0. OASIS XACML-TC, Committee Draft 04, December 2004.
23. A. Tripathi and N. Karnik. Protected resource access for mobile agent-based distributed computing. In *Proceedings of the ICPP workshop on Wireless Networking and Mobile Computing*, 1998.
24. J. Vollbrecht, P. Calhoun, S. Farrell, L Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA Authorization Framework. RFC-2904, The Internet Society, August 2000.